Une plate-forme pour le test de cartes hybrides

Bertrand GILLES Valérie-Anne NICOLAS Lionel MARCÉ

EA2215 - LIMI / DÉPARTEMENT INFORMATIQUE
Université de Bretagne Occidentale
20 avenue Le Gorgeu - BP 809 - 29285 Brest Cédex, France
E-mail :{gilles|vnicolas|marce}@univ-brest.fr

1 Introduction

Le test est une activité qui intervient tout au long du cycle de vie d'un composant sous différentes formes : analyse de testabilité lors de la phase de conception puis, que ce soit en phase de développement/production ou de maintenance, conception et génération de données de test, exécution des données de test et enfin interprétation des résultats obtenus [12]. Le développement des technologies multi-média et des télécommunications, l'utilisation croissante de systèmes embarqués (transports, téléphonie mobile...) requièrent des composants matériels hybrides numériques/analogiques de plus en plus sûrs, et donc un processus de test adapté, fiable et robuste. Par ailleurs, l'intégration grandissante des composants (technologies submicroniques) et la diminution régulière des délais de conception / production sont autant de contraintes technologiques nouvelles pour le processus de test.

Dans ce contexte, il est crucial de mieux maîtriser les phases de test et de maintenance des composants ou cartes hybrides (incluant le diagnostic et la correction des défauts). Notre travail se situe dans ce cadre et concerne plus particulièrement le test en maintenance de cartes hybrides. En nous appuyant sur une étude de cas, nous proposons une méthode de test de cartes hybrides. Les spécificités du test en maintenance nous ont conduits à adopter une approche fonctionnelle, que ce soit au niveau modélisation de la carte ou au niveau du choix des stratégies de test.

Nous commençons par présenter notre approche (modélisation et validation d'une carte hybride) en section 2, puis un prototype préliminaire permettant de la mettre en oeuvre en section 3. Nous détaillons ensuite en section 4 l'application de la méthode proposée à une étude de cas issue du monde industriel. Nous terminons par une discussion sur les limites et les perspectives de cette approche.

2 Présentation de la méthode de test de cartes hybrides

Une carte hybride est constituée de composants analogiques et de composants numériques. Les cartes hybrides que nous désirons tester (cf. section 4) ne font intervenir les composants analogiques que dans les interfaces (d'entrée, de sortie) de la carte. Les composants numériques constituent quant à eux le coeur de la carte, la partie contrôle. Ce type de cartes correspond à une part importante des cartes hybrides, utilisées notamment dans les applications audio, vidéo, de surveillance/contrôle de capteurs, etc. La méthode que nous présentons ici a pour but de tester spécifiquement ce type de cartes hybrides.

Dans le domaine du test matériel, la conception des données de test est guidée par les caractéristiques physiques (seuils, tolérances) et les performances attendues des composants. Cela est grandement facilité par l'utilisation de composants auto-testables dans l'élaboration des cartes. Ce type de test est tout fait adapté pour le test en production (verdict attendu de type : conforme / non conforme) mais n'est plus suffisant pour le test en maintenance. La spécificité du test en maintenance est en effet de vérifier que certaines fonctionnalités de la carte ont le comportement attendu, sans pouvoir se baser sur une liste de fautes préétablie (limitation dûe à l'aspect analogique). Le test doit alors être basé sur des critères fonctionnels et non structurels. Un jeu de test permettra ainsi de valider le comportement d'une carte par rapport à sa spécification fonctionnelle.

Nous présentons ci-après les différentes étapes constitutives de notre méthode : la modélisation fonctionnelle d'une carte hybride, puis sa validation via des procédures de test fonctionnel et/ou l'utilisation d'outils de vérification.

2.1 Modélisation fonctionnelle d'une carte hybride

Les cartes hybrides que nous désirons tester ne font intervenir les composants analogiques que dans les interfaces de la carte, les composants numériques constituant exclusivement la partie contrôle de la carte. Malgré la coexistence de composants analogiques et numériques, ce type de carte n'est pas ce qu'on appelle à proprement parler un système hybride, car il n'y a pas de dynamique mixte discrète-continue sous-jacente. En effet, le système ainsi construit suit une dynamique continue dans ses interfaces et une dynamique discrète dans sa partie contrôle, sans qu'il y ait d'interférences entre ces deux dynamiques. Bien qu'ils offrent un cadre uniforme, il ne nous a donc pas paru adéquat d'utiliser des formalismes mixtes (comme les automates hybrides [1] ou la logique MLD [3] spécifiques aux systèmes hybrides) pour modéliser ce type de carte. En effet, les formalismes mixtes mettent l'accent sur la modélisation de l'entrelacement des dynamiques discrète et continue, ce qui ne présente pas d'intérêt particulier dans le cas présent.

Nous avons par conséquent opté pour des modélisations relativement indépendantes des parties contrôle et interfaces. Notre démarche est la suivante : modélisation fonctionnelle des aspects numériques d'une part, modélisation fonctionnelle des aspects analogiques d'autre part, modélisation fonctionnelle globale de la carte en définissant un cadre commun dans lequel exprimer et faire interagir les deux modélisations précédentes.

La partie contrôle se modélise assez bien de manière discrète, sous forme d'automate, et plus précisemment en tant que système séquentiel produisant des sorties (machine de Mealy).

Pour la modélisation des interfaces (composantes analogiques), nous avons consideré deux possibilités, l'une dans le domaine continu, l'autre dans le domaine discret : soit modéliser directement une fonction analogique par des équations différentielles ; soit abstraire la fonctionnalité analogique (par rapport au temps) puis modéliser par une machine de Mealy.

La première proposition permet de conserver le caractère continu de l'analogique, mais introduit un formalisme différent des automates de la partie contrôle. Cela signifie que pour construire la modélisation globale de la carte, il faudra faire communiquer entre eux ces deux formalismes, ce qui n'est pas chose aisée à moins d'utiliser un formalisme mixte comme ceux cités en début de section.

La seconde proposition facilite la modélisation globale de la carte : un ensemble d'automates communiquants. Cependant il est nécessaire d'abstraire la fonctionnalité analogique par rapport au temps, c'est-à-dire la numériser. La question importante dans ce cas est donc de savoir si cette abstraction conserve suffisamment des caractéristiques de la fonction analogique pour que les processus de test ou de vérification consécutifs à la modélisation soient valides et pertinents [2].

Pour choisir entre ces deux propositions, il est nécessaire de réfléchir plus précisemment à la représentation des données (entrées/sorties de la carte hybride) la plus appropriée dans le cadre de notre méthode. Les entrées/sorties d'une carte hybride sont des signaux électriques continus ou numériques (caractéristiques de tensions, de courants, de fréquences).

Dans le cas de la modélisation d'une fonction analogique par des équations différentielles, un signal sera lui aussi représenté par une équation différentielle.

Dans le cas de la modélisation par une machine de Mealy après abstraction par rapport au temps de la fonction analogique, la modélisation d'un signal sera une abstraction par rapport au temps du signal réel, c'est-à-dire un échantillonnage du signal.

Pour notre objectif de génération de données de test, une représentation discrète par échantillonnage du signal parait plus appropriée car plus facilement manipulable. On peut ainsi également bénéficier de bibliothèques de fonctions de discrétisation pour diverses fonctions analogiques. De plus, ce choix prend encore plus de sens dans la perspective d'une approche qualitative [10] (ou symbolique) du test, ce qui est souvent le cas pour le test de composants purement analogiques.

En tenant compte des arguments exposés précédemment, la modélisation de carte hybride que nous avons finalement choisie consiste à modéliser une carte par un ensemble d'automates (machines de Mealy) communiquants et un ensemble de signaux abstraits d'entrées/sorties. Étant donné que nous travaillons sur un ensemble d'automates communiquants, les transitions de ces automates seront de la forme: «condition de synchronisation $\rightarrow$ garde booléenne». La sémantique associée est la suivante : quand la condition de synchronisation est vérifiée, on évalue la garde booléenne, et si celle-ci est vraie alors on franchit la transition en exécutant l'action associée (la «sortie») si elle existe.

La partie contrôle est ainsi modélisée directement par une machine de Mealy. Les parties analogiques sont abstraites puis modélisées par des machines de Mealy. Un signal abstrait est le résultat d'un échantillonnage du signal réel.

Ce choix de modélisation est correct sous réserve que l'abstraction utilisée soit suffisante et valide par rapport aux besoins en termes de test et de vérification de la carte. Pour l'étude de cas présentée en section 4, cette modélisation s'est averée satisfaisante.

2.2 Test fonctionnel de cartes hybrides

Comme nous l'avons déjà dit, les spécificités du test en maintenance nous ont conduits à adopter une approche fonctionnelle, que ce soit au niveau modélisation de la carte ou au niveau du choix des stratégies de test.

Après avoir défini la notion de cas de test, nous présentons les différentes stratégies de test que nous proposons pour répondre aux besoins du test en maintenance.

2.2.1 Définition d'un cas de test

Un cas de test est composé de :

- conditions de contrôlabilité ;
- conditions d'observabilité ;
- caractérisations de données de test ;
- résultats attendus.

La *caractérisation d'une donnée de test* correspond à une représentation symbolique (non instanciée) de la donnée. Cela peut prendre la forme d'une énumération, d'une représentation ensembliste ou, dans notre cas, d'un système de contraintes sur la donnée. Il est important de travailler sur une représentation symbolique des données pour la validité et l'efficacité de l'approche. L'instanciation des données de test n'est réalisée qu'au final, sur demande de l'utilisateur, pour fournir des données réelles à un banc de test par exemple.

Les *résultats attendus* peuvent être des sorties (affichages, écritures dans des registres ou en mémoire, erreurs...) mais aussi les séquences des états ou transitions parcourus au niveau modélisation.

Les *conditions de contrôlabilité et d'observabilité* peuvent paraître inutiles car au niveau modélisation (machines de Mealy), tous les états et transitions sont observables et contrôlables. Il faut cependant garder en mémoire que si l'on génère un cas de test sur la modélisation, on l'exécute au final sur la carte réelle dont la contrôlabilité et l'observabilité sont habituellement beaucoup plus restreintes. Les conditions de contrôlabilité doivent donc permettre de se positionner dans une configuration adaptée à une exécution correcte d'un cas de test. De même, les conditions d'observabilité doivent permettre de se positionner dans une configuration où les résultats attendus, significatifs du succès ou de l'échec du test, sont effectivement observables. En pratique, il est courant d'inclure les données de test dans des séquences de test, permettant ainsi d'aller d'une configuration contrôlable à une configuration observable en passant par le comportement à tester.

2.2.2 Stratégies de test fonctionnel pour le test en maintenance

D'un point de vue fonctionnel général, le testeur cherche à répondre aux questions suivantes :

1. Quel est l'ensemble des comportements possibles ?
2. L'ensemble des comportements modélisés correspond-il à l'ensemble des comportements souhaités ?
3. Un comportement donné de la carte est-il possible ? Si oui, comment le reproduire ?
4. Peut-on être sûr qu'un comportement donné est impossible ?

Bien que l'on puisse répondre à certaines de ces questions par le biais de la vérification, ce n'est généralement pas l'approche choisie par les industriels lors des phases de maintenance. Aussi nous concentrons nous sur les différentes stratégies de test applicables dans ce domaine.

Dans notre modélisation, les différents comportements de la carte sont représentés par l'ensemble des chemins possibles dans un automate (machine de Mealy). De nombreux travaux ont porté sur la génération de données de test à partir d'automates [11], [7]. Une stratégie de test consistant à parcourir toutes les combinaisons d'états et de transitions, et à générer un cas de test associé à chacune de ces combinaisons, répond à la première question. En comparant cet ensemble de cas de test avec une spécification de la carte, on répond alors à la deuxième question. Pour la troisième question, il suffit de repérer dans l'automate le chemin correspondant au comportement recherché, et à essayer de le sensibiliser. Le résultat de cette sensibilisation, un cas de test, permettra de reproduire le comportement par la suite. Si la sensibilisation échoue, cela signifie qu'au-moins un des états du chemin étudié est inaccessible, et que le comportement associé ne se produira jamais (réponse à la dernière question).

Dans le cadre du test en maintenance, les stratégies énoncées ci-dessus ne sont pas suffisantes. En effet, on entre en phase de maintenance le plus souvent à la suite d'une panne, d'un comportement erroné de la carte. Pour corriger la panne, il faut localiser le défaut à son origine. Il est donc nécessaire de disposer de stratégies de test local à un comportement et non pas global comme les précédentes. Ces stratégies de test local doivent notamment permettre de :

- passer une fois dans chaque état ;
- passer une fois par chaque transition ;
- tester toutes les transitions (*) ;

- tester tous les états (**) ;
- réaliser des tests paramétrables par l'utilisateur avec focalisation sur certains états, transitions, ou combinaisons d'états et de transitions.

(*) Le test d'une transition est réalisé par un ensemble de cas de test permettant d'exercer la transition pour ses valeurs nominales et pour ses valeurs limites.

(**) Le test d'un état est réalisé par un ensemble de cas de test exerçant toutes les combinaisons possibles de transitions entrantes et sortantes autour de cet état.

3 Plate-forme de test

La méthode présentée en section 2 offre un cadre homogène pour la modélisation fonctionnelle, le test fonctionnel et la vérification de propriétés de cartes hybrides. D'un point de vue mise en oeuvre, cela se traduit par une plate-forme ouverte et homogène permettant la modélisation, le test et la vérification de cartes hybrides. Un prototype de cette plate-forme est en cours de réalisation. Actuellement, il permet de modéliser une carte en termes d'automates (machines de Mealy). Il permet également de concevoir et générer des données de test selon les différentes stratégies de test fonctionnel local ou global vues à la section 2.2.

À terme, la plate-forme comportera un module de discrétisation/reconstitution. Ce module permettra d'instancier les données de test par des valeurs réelles du domaine d'application de la carte, en se basant sur la modélisation de ses entrées. Dans un souci de complémentarité, la plate-forme offrira aussi des interfaces directes (via des modules de traduction automatique) vers des outils de vérification ou simulation comme Kronos [5] ou Uppaal [9]. De même, elle comportera un module de traduction automatique de notre formalisme de modélisation vers le langage VHDL-AMS. Le langage VHDL-AMS est en effet un standard incontournable dans toute la chaîne de production de cartes. On le retrouve notamment comme format de modélisation dans les logiciels de CAO, ainsi qu'en format d'entrée de la plupart des simulateurs et bancs de test.

Le prototype a été élaboré en suivant une conception objet et en utilisant UML (Unified Modeling Language). Ce type de conception est très générique et fait de notre prototype un logiciel ouvert, et donc évolutif. Le prototype est écrit en C++ et s'appuie sur le framework graphique ILOG Views. Ce framework est une librairie C++ de construction d'interfaces graphiques 2D homme-machine. ILOG Views est portable et supporte un nombre important de plate-formes. Actuellement, le prototype est développé et s'exécute sur un PC sous Linux équipé du système d'exploitation RedHat 7.2.

La partie concernant la mise en oeuvre des stratégies de test est réalisée dans un formalisme logique et en utilisant la programmation par contraintes, avec le solveur ECL^{i}PS^e [4]. La programmation par contraintes est très pertinente pour la génération de données de test [6], [8], car elle permet de travailler sur des plages de valeurs de test dans un premier temps, que le testeur peut ensuite instancier à sa guise, en particulier pour prendre en compte des contraintes applicatives ou environnementales particulières. Par ailleurs, ce type de programmation très souple au niveau du mode des arguments pourrait permettre d'utiliser ces mêmes algorithmes de test dans un but de simulation. Ainsi, la plate-forme pourrait non seulement générer des données de test mais aussi simuler leur exécution sur la modélisation. Cela constituerait un complément appréciable lors de l'interprétation des résultats du test ainsi qu'une aide au diagnostic.

4 Application à une étude de cas

Ce projet bénéficie d'une coopération avec l'antenne brestoise de la société ISIS-MPP, spécialisée dans le test et l'outillage pour le test de cartes électroniques. Cette société nous a ainsi proposé de mener une étude de cas sur une de leurs cartes hybrides : la carte Tachy. Nous détaillons à présent le traitement de cette carte selon la méthode présentée en section 2.

4.1 Modélisation de la carte Tachy

4.1.1 Description de la carte Tachy

La carte Tachy est une carte hybride qui possède à la fois des fonctions analogiques et numériques. Cette carte possède quatorze voies analogiques recevant les signaux de génératrices tachymètriques et des sorties numériques appartenant essentiellement à un bus VME. Le signal analogique émis par une génératrice est un signal carré dont la fréquence est proportionnelle à sa vitesse de rotation.

La fonction globale de la carte est de surveiller cycliquement les entrées génératrices par comparaison par rapport à des seuils (de tension), et d'enregistrer en mémoire RAM le numéro des voies en défaut. Un défaut peut représenter une absence de signal tachymètrique ou un franchissement des seuils.

4.1.2 Modélisation des entrées

Les entrées de la carte Tachy correspondent aux signaux émis respectivement par les quatorze génératrices tachymètriques. Chacun de ces signaux est

reçu sur une voie dédiée. La carte numérise (discrétise en temps et en amplitude) le signal analogique de la $i^{\text{ème}}$ génératrice reçu sur la $i^{\text{ème}}$ voie afin que la partie numérique de la carte puisse l'exploiter ($1 \leq i \leq 14$).

Les quatorze signaux sont modélisés de la même manière. La modélisation consiste à remonter plus en amont la numérisation du signal analogique, et en fin de compte, à substituer au signal analogique sa version numérisée. Ansi, le signal analogique d'une voie donnée est modélisé par une succession d'échantillons espacés dans le temps de la période d'échantillonnage utilisée.

4.1.3 Modélisation de la fonction analogique - interface d'entrée

C'est une fonction analogique qui consiste à faire suivre les quatorze signaux en entrée vers la partie contrôle à travers quatorze voies analogiques indépendantes.

Chaque voie analogique V_i, $1 \leq i \leq 14$, est modélisée par un automate A_i composé d'un unique état (initial) nommé EV_i, et d'une unique transition de EV_i vers lui-même, étiquetée Ctrl!x.

La notation Ctrl!x signifie l'envoi de la donnée x par l'automate A_i vers l'automate Ctrl modélisant la partie contrôle de la carte. La donnée x représente un échantillon numérique du signal modélisé, lu sur la voie V_i.

4.1.4 Modélisation de la partie contrôle

La partie contrôle de la carte, c'est-à-dire sa partie numérique, est modélisée par un automate Ctrl. Cet automate comprend quatorze états nommés LV_i avec $1 \leq i \leq 14$, LV_1 étant l'état initial. De plus, chaque état LV_i, $1 \leq i \leq 13$, dispose de trois transitions vers un état LV_{i+1}, et l'état LV_{14} de trois transitions vers l'état LV_1.

Les transitions d'un état LV_i vers un état LV_{i+1} , $1 \leq i \leq 13$, (ou de LV_{14} vers LV_1) sont les suivantes :

- $A_i?x \rightarrow (x \geq s_i\text{inf} \;\&\&\; x \leq s_i\text{sup})$: l'automate Ctrl passe de l'état LV_i à l'état LV_{i+1} sur réception de la donnée x provenant de l'automate A_i. La valeur de x appartient à l'ensemble des valeurs autorisées determiné par les seuils inférieur ($s_i$inf) et supérieur ($s_i$sup) associés à la voie V_i. C'est le cas nominal, il n'y a pas d'action associée à cette transition.

- $A_i?x \rightarrow x < s_i\text{inf}$: l'automate Ctrl passe de l'état LV_i à l'état LV_{i+1} sur réception de la donnée x provenant de l'automate A_i. La valeur de x est incorrecte car elle est inférieure au seuil inférieur ($s_i$inf) de la voie V_i. L'action associée est un enregistrement en mémoire du numéro de la voie en défaut (i).

- $A_i?x \rightarrow x > s_i\text{sup}$: l'automate Ctrl passe de l'état LV_i à l'état LV_{i+1} sur réception de la donnée x provenant de l'automate A_i. La valeur de x est incorrecte car elle est supérieure au seuil supérieur ($s_i$sup) de la voie V_i. L'action associée est un enregistrement en mémoire du numéro de la voie en défaut.

4.2 Test de la carte Tachy

Pour des raisons de concision, nous n'illustrons ici que les deux stratégies de test à la base de celles présentées en section 2.2, à savoir le test d'une transition et le test d'un état.

4.2.1 Test de la transition nominale de l'état $\mathbf{Lv_1}$ vers l'état $\mathbf{Lv_2}$

Pour ce test, trois données de test (DT_i, $1 \leq i \leq 3$) sont nécessaires. Elles sont listées dans le tableau 1. Chaque DT_i est une valeur x soumise à une contrainte CDT_i. Chaque DT_i a comme condition de contrôlabilité: «être dans l'état Lv_1», et comme condition d'observabilité : «être dans l'état Lv_2».

DT	CDT	Résultat prévu
DT_1	$(x > s_1\text{inf}) \wedge (x < s_1\text{sup})$	Pas d'erreur
DT_2	$x = s_1\text{inf}$	Pas d'erreur
DT_3	$x = s_1\text{sup}$	Pas d'erreur

TAB. 1 - *Test de la transition nominale de l'état Lv_1 vers l'état Lv_2*

4.2.2 Test de l'état $\mathbf{Lv_2}$

Pour ce test, neuf données de test (DT_i, $1 \leq i \leq 9$) sont nécessaires. Elles sont listées dans le tableau 2. Chaque DT_i est une séquence de deux valeurs de la forme (x_1;x_2) soumise à une contrainte CDT_i. Chaque DT_i a comme condition de contrôlabilité: «être dans l'état Lv_1», et comme condition d'observabilité : «être dans l'état Lv_3».

5 Conclusion et travaux futurs

Nous avons présenté une méthode ainsi qu'une architecture et les premières briques d'une plate-forme pour le test de cartes hybrides. Nous n'en sommes qu'au stade initial et le chemin est encore long avant de pouvoir valider cette méthode. Certains points nous semblent cependant acquis, comme l'approche fonctionnelle à tous les niveaux, le formalisme homogène de modélisation et les stratégies de test exposées précédemment.

À court terme, il est nécessaire de compléter et enrichir le prototype de la plate-forme par des interfaces vers certains outils de vérification ; des générateurs de données analogiques (c'est-à-dire des mises

DT	CDT	Résultat attendu
DT_1	$(x_1 \geq s_1\text{inf}) \wedge (x_1 \leq s_1\text{sup}) \wedge (x_2 \geq s_2\text{inf}) \wedge (x_2 \leq s_2\text{sup})$	Pas d'enregistrement d'erreur
DT_2	$(x_1 \geq s_1\text{inf}) \wedge (x_1 \leq s_1\text{sup}) \wedge (x_2 < s_2\text{inf})$	erreur(inf,2)
DT_3	$(x_1 \geq s_1\text{inf}) \wedge (x_1 \leq s_1\text{sup}) \wedge (x_2 > s_2\text{sup})$	erreur(sup,2)
DT_4	$(x_1 < s_1\text{inf}) \wedge (x_2 \geq s_2\text{inf}) \wedge (x_2 \leq s_2\text{sup})$	erreur(inf,1)
DT_5	$(x_1 < s_1\text{inf}) \wedge (x_2 < s_2\text{inf})$	erreur(inf,1), erreur(inf,2)
DT_6	$(x_1 < s_1\text{inf}) \wedge (x_2 > s_2\text{sup})$	erreur(inf,1), erreur(sup,2)
DT_7	$(x_1 > s_1\text{sup}) \wedge (x_2 \geq s_2\text{inf}) \wedge (x_2 \leq s_2\text{sup})$	erreur(sup,1)
DT_8	$(x_1 > s_1\text{sup}) \wedge (x_2 < s_2\text{inf})$	erreur(sup,1), erreur(inf,2)
DT_9	$(x_1 > s_1\text{sup}) \wedge (x_2 > s_2\text{sup})$	erreur(sup,1), erreur(sup,2)

TAB. 2 - *Test de l'état Lv_2*

en oeuvre de fonctions de reconstitution) ; un traducteur automatique vers le langage VHDL/AMS. Ce dernier développement permet d'assurer la compatibilité avec les interfaces des testeurs matériels (numériques, analogiques), des ateliers de conception de cartes, des outils de vérification/simulation.

À plus long terme, il faudrait lever la restriction actuelle sur le type de carte, dûe à l'étude de cas sur laquelle nous travaillons, afin de proposer des traitements spécifiques aux différents types de cartes (sous forme de bibliothèques par exemple).

Au final, ce travail de formalisation de l'activité de test de cartes hybrides doit conduire à des outils automatiques (ou semi-automatiques) de gestion des phases de test et maintenance, spécialisables par rapport à des types de composants/cartes précis. Ce type d'outils répondrait à un réel besoin du monde industriel.

Remerciements

Nous tenons à remercier Michel Le Goff, de la société ISIS-MPP, pour sa disponibilité et les nombreux éclaircissements qu'il a bien voulu nous communiquer sur la pratique industrielle du test de cartes hybrides. Nous le remercions aussi de nous avoir fourni la carte Tachy comme base pour notre étude cas.

Références

[1] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. *Hybrid Systems I, Lecture Notes in Computer Science*, 736:209–229, 1993. Springer-Verlag.

[2] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. In *Proceedings of the IEEE*, volume 88, pages 971–984, 2000.

[3] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:407–427, 1999.

[4] A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace. Eclipse: An introduction. Technical Report IC-Parc-03-1, IC-Parc, Imperial College London, 2003.

[5] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. *Hybrid Systems III, Lecture Notes in Computer Science*, 1066:208–219, 1996. Springer-Verlag.

[6] R. A. Demillo and A. J. Offutt. Constraint-Based Automatic Test Data Generation. *IEEE Transactions on Software Engineering*, 17(9):900–910, September 1991.

[7] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test Selection Based on Finite State Models. *IEEE Transactions on Software Engineering*, 17(6):591–603, June 1991.

[8] B. Korel. Automated Software Test Data Generation. *IEEE Transactions on Software Engineering*, 16(8):870–879, August 1990.

[9] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1(1), 1997.

[10] L. Travé-Massuyès, P. Dague, and F. Guerrin. *Le raisonnement qualitatif pour les sciences de l'ingénieur*. Hermès, 1997.

[11] G. von Bochmann and A. Petrenko. Protocol Testing: Review of Methods and Relevance for Software Testing. *ISSTA*, pages 109–124, August 1994. ACM.

[12] S. Xanthakis, M. Maurice, A. De Amescua, O. Houri, and L. Griffet. *Test & Contrôle des Logiciels: Méthodes, Techniques & Outils*. EC2, 1994.